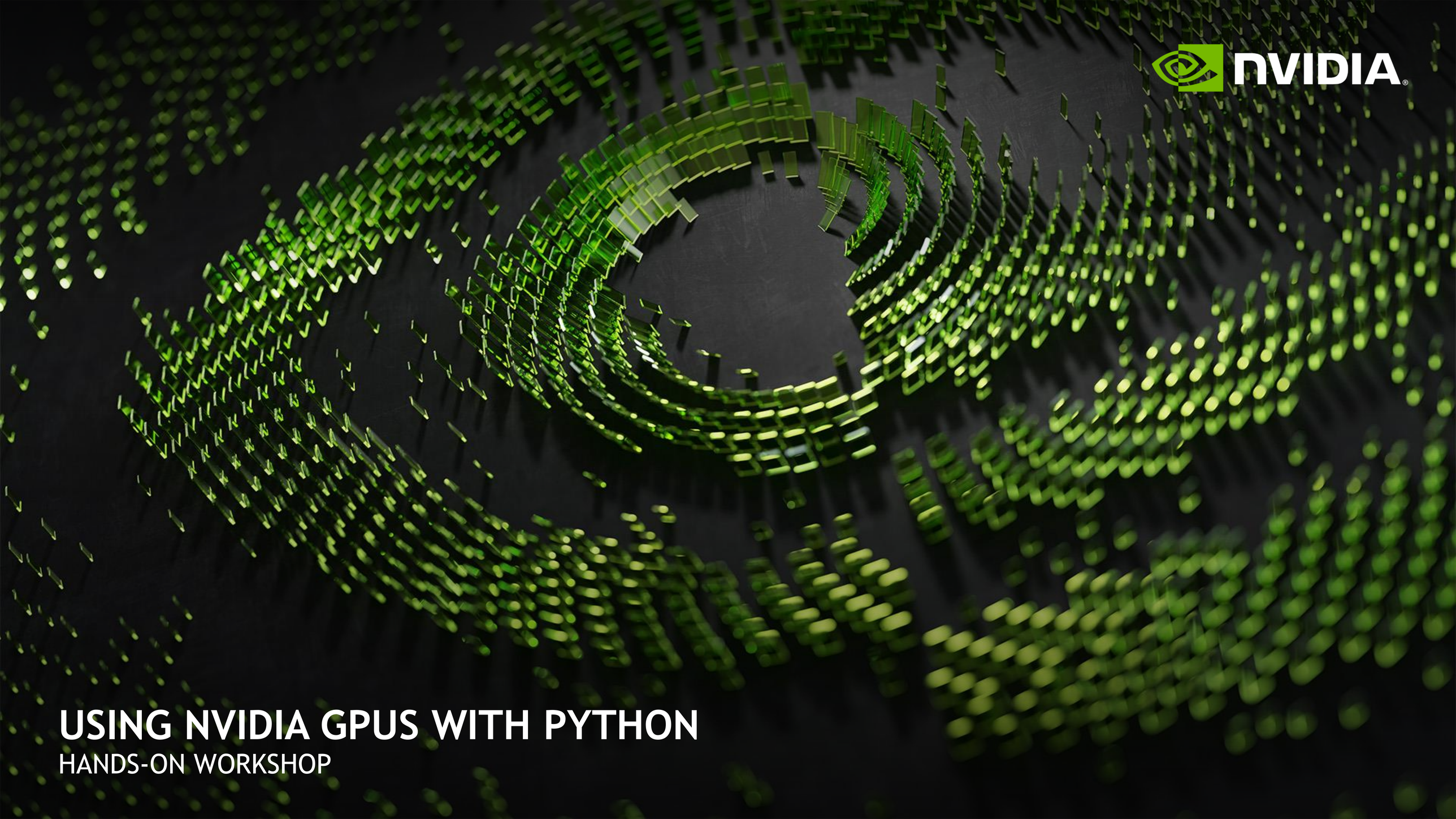


USING NVIDIA GPUS WITH PYTHON

HANDS-ON WORKSHOP

AGENDA

Accessing the Workshop Materials

Overview of GPU Computing & Python Ecosystem

GPU-Accelerated Numerical Computing with ***CuPy***

GPU-Accelerated Data Manipulation with ***RAPIDS cuDF***

GPU-Accelerated Data Science with ***RAPIDS cuML***

GPU-Accelerated Differentiable Spatial Computing with *Warp*

Reading Section:
"Ecosystem Overview"

KEY COMPONENTS OF THE GPU ACCELERATED PYTHON ECOSYSTEM

Kernel-Generation & Numerical Computing

- NVMath Python
- Warp
- Numba
- cuda-python

Data Processing Libraries & Machine Learning

- Dask
- RAPIDS
- CuPy
  - cuDF
  - cuML
  - cuGraph
- SciKit-Learn
- XGBoost

Learning Tools

- Jupyter-Notebooks
- Anaconda / Conda-Forge

Model Serving

- Triton Inference Server

GPU Accelerated Inference

- vLLM
- TensorRT
- TensorRT-LLM
- LMDeploy

REPLICATING THE WORKSHOP ENVIRONMENT

NVIDIA NGC
RAPIDS Container
https://ngc.nvidia.com

RAPIDS
Release Selector
https://rapids.ai

Containers › RAPIDS Base

RAPIDS Base

Get Container

RAPIDS

Accelerated with NVIDIA

Features

NVIDIA AI Enterprise Supported

Description

The RAPIDS suite of software libraries gives you the freedom to execute end-to-end data science and analytics pipelines entirely on GPUs.Please add description

Publisher

NVIDIA

Latest Tag

24.12-cuda12.5-py3.12

Modified

January 31, 2025

Compressed Size

5.2 GB

Multinode Support

No

Multi-Arch Support

Yes

Overview | Tags | Layers | Security Scanning | Related Collections

RAPIDS - Open GPU Data Science

What is RAPIDS?

Visit rapids.ai for more information.

The RAPIDS suite of software libraries gives you the freedom to execute end-to-end data science and analytics pipelines entirely on GPUs. It relies on NVIDIA® CUDA® primitives for low-level compute optimization, but exposes GPU parallelism and high-bandwidth memory speed through user-friendly Python interfaces.

NOTE: Review our system requirements to ensure you have a compatible system!

Current Version - RAPIDS v24.12

RAPIDS Libraries included in the images:

- cuDF
- cuML
- cuGraph
- cuVS
- RMM
- RAFT
- cuSpatial
- cuxfilter
- cuCIM

RELEASE	Stable (24.12)	Nightly (25.02a)					
METHOD	Conda	pip	Docker				
ENV. CUDA	CUDA 11.4 - 11.8	CUDA 12.0 - 12.5					
PYTHON	Python 3.10	Python 3.11	Python 3.12				
RAPIDS PACKAGES	Standard	Choose Specific Packages					
ADDITIONAL PACKAGES	Graphistry	JupyterLab	NetworkX + nx-cugraph	Plotly Dash	PyTorch	TensorFlow	Xarray-Spatial

RAPIDS currently doesn't support `channel_priority: strict`; use `channel_priority: flexible` instead

The `Standard` selection contains the following packages:

`cuDF`, `cuML`, `cuGraph`, `nx-cugraph`, `cuSpatial`, `cuProj`, `cuxfilter`, `cuCIM`, `RAFT`, `cuVS`

COMMAND

```
conda create -n rapids-24.12 -c rapidsai -c conda-forge -c nvidia  \
    rapids=24.12 python=3.12 'cuda-version>=12.0,<=12.5'
```

COPY COMMAND

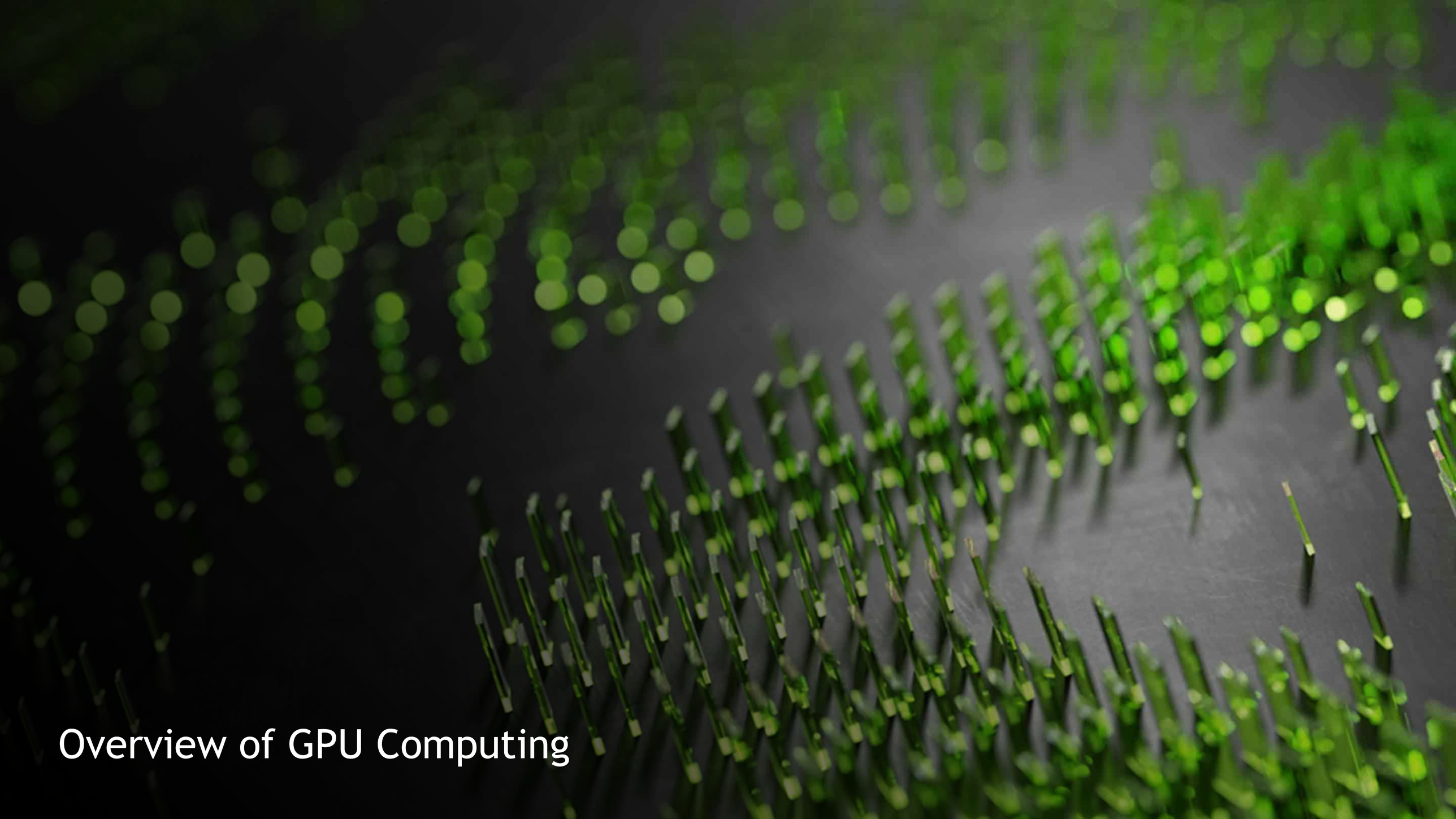
https://github.com/NVIDIA/warp

pip install warp-lang

NVIDIA

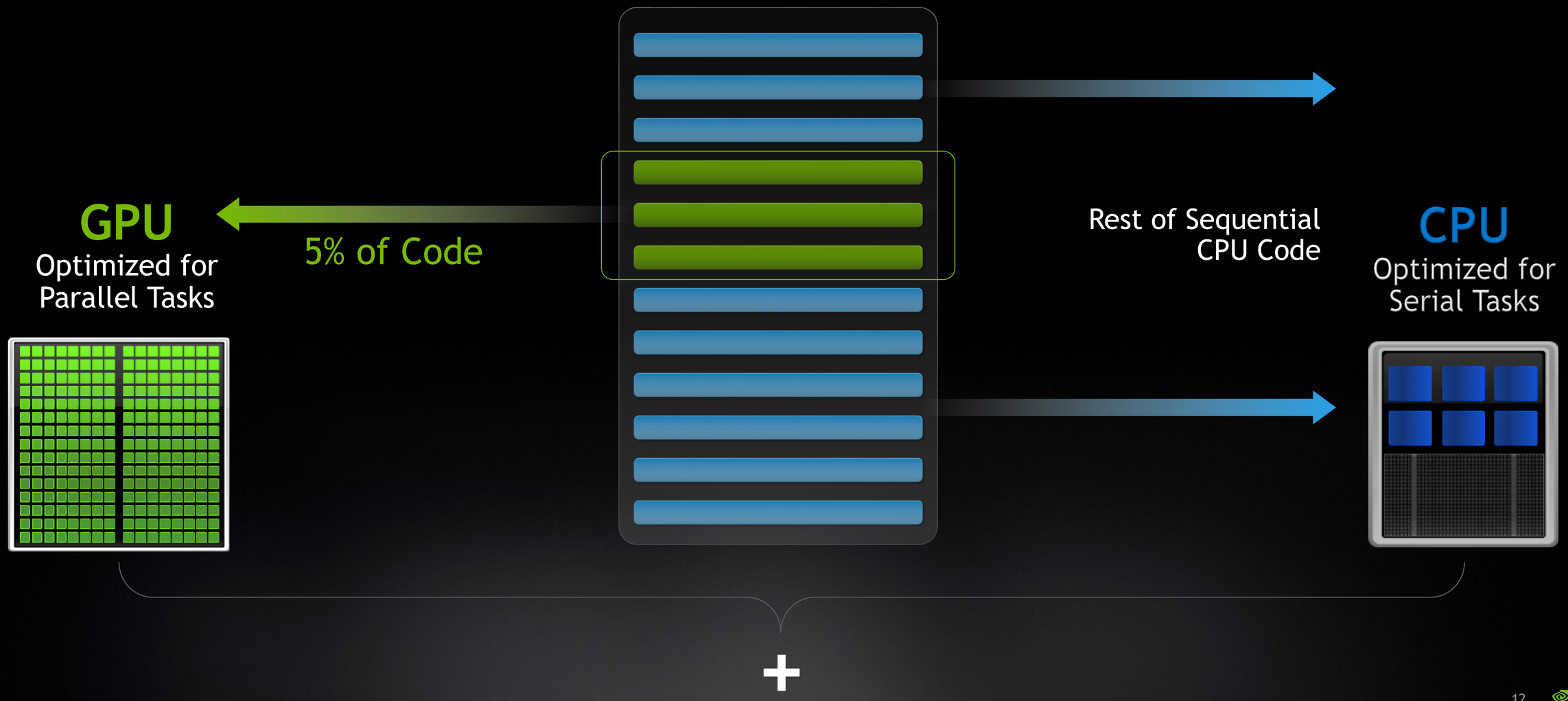
Notebook 0: The Accelerated Python Ecosystem

Working in Section:
"Introduction to Workshop Lab Environment"



Overview of GPU Computing

ACCELERATED COMPUTING WITH GPUS



This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

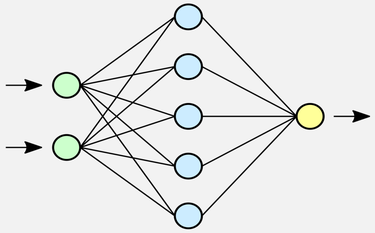
THE GAP BETWEEN PYTHON AND GPUS

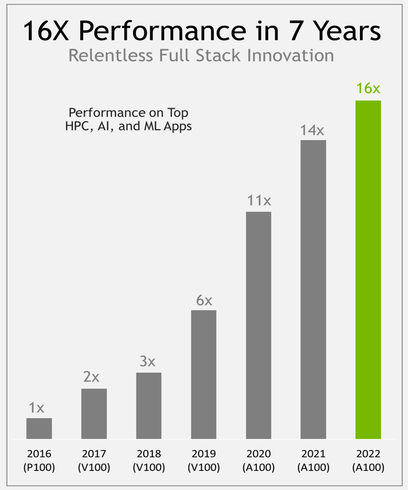
Python

- High-level
- Interactive, versatile
- Easy to prototype
- Automatic memory management
- Dynamic typing
- Robust package management (conda, pip)
- MANY libraries available

NVIDIA GPUs

- C, Fortran
- Compiled
- High-Performance



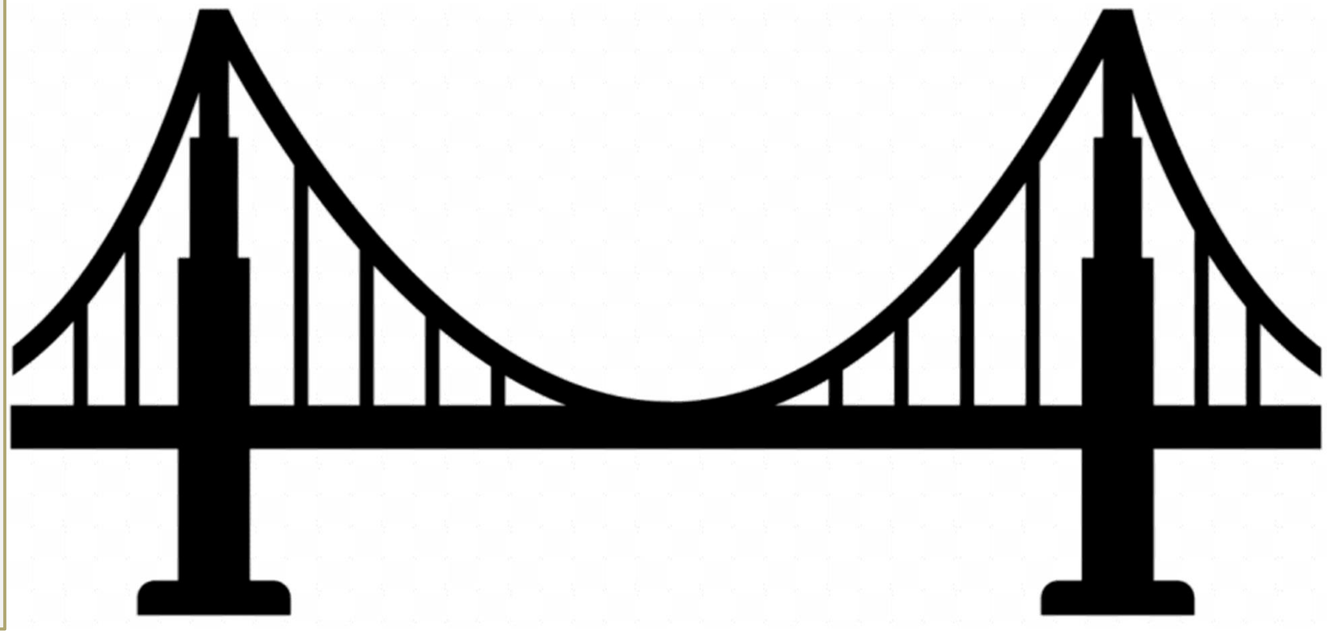


BRIDGING THE GAP BETWEEN PYTHON AND GPUS

Python

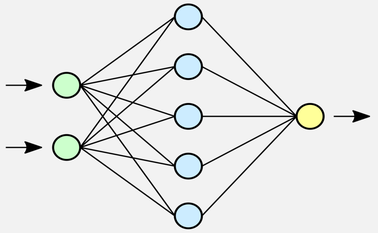
- High-level
- Interactive, versatile
- Easy to prototype
- Automatic memory management
- Dynamic typing
- Robust package management (conda, pip)
- MANY libraries available

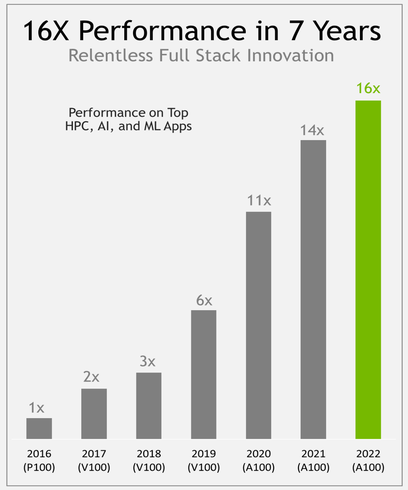
JIT Compilation → Low-Level Language Bindings → CUDA-X Libraries



NVIDIA GPUs

- C, Fortran
- Compiled
- High-Performance





A FEW GENERAL TIPS FOR SUCCESSFUL GPU COMPUTING

Minimize data movement to and from the GPU

- What happens on the GPU, stays on the GPU!
- PCI express is a bottleneck for data movement
- Try NVLink for GPU peer-to-peer, 600 GB/s!

GPUs are parallel processing machines

- Leave serial operations to the CPU
- Look for high arithmetic intensity, chunky loops, dense linear algebra
- Experiment with reduced precision, mixed-precision iterative refinement
- High memory bandwidth - Fast FFTs.

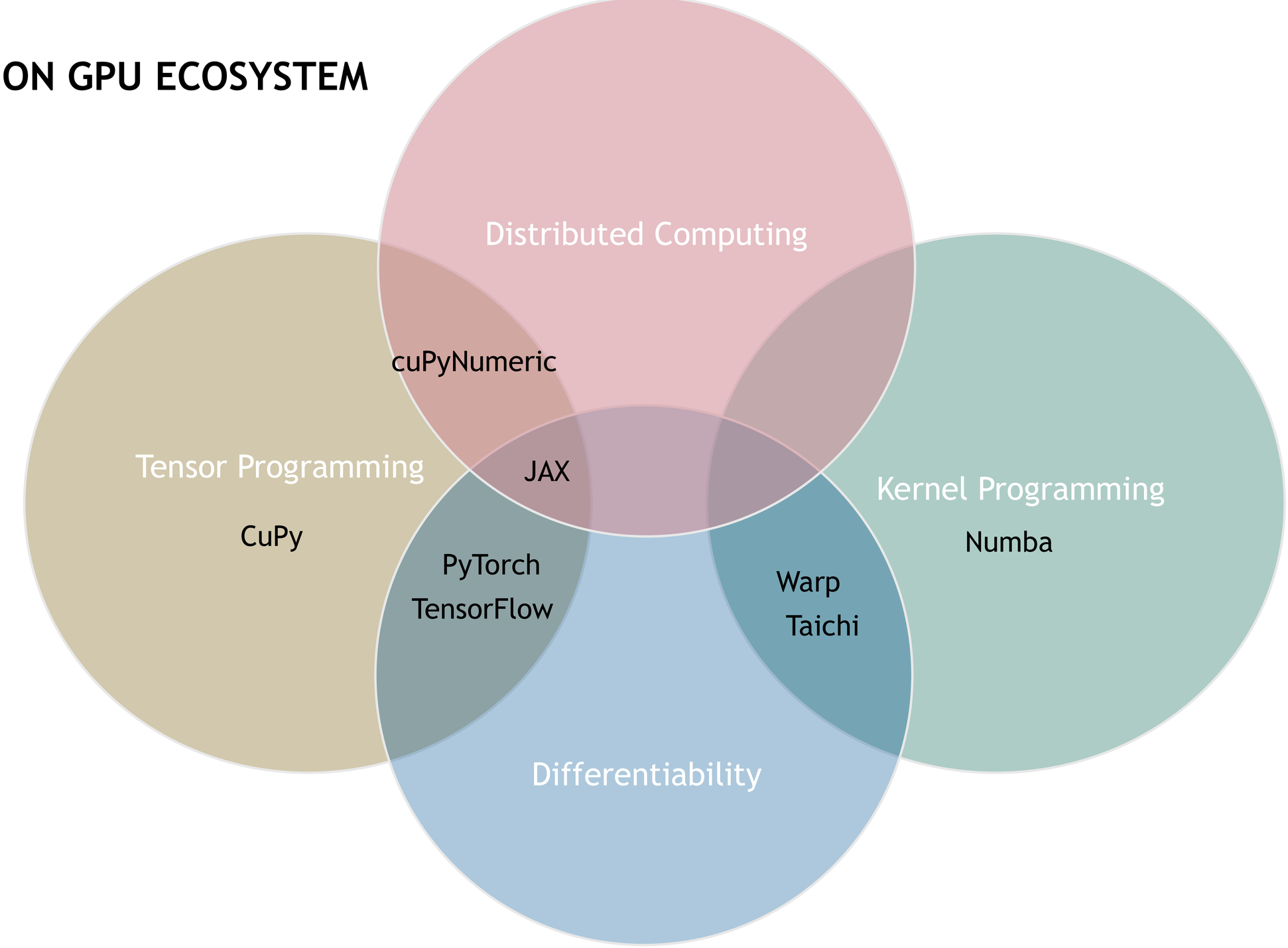
Stand on the Shoulders of Those Before You!

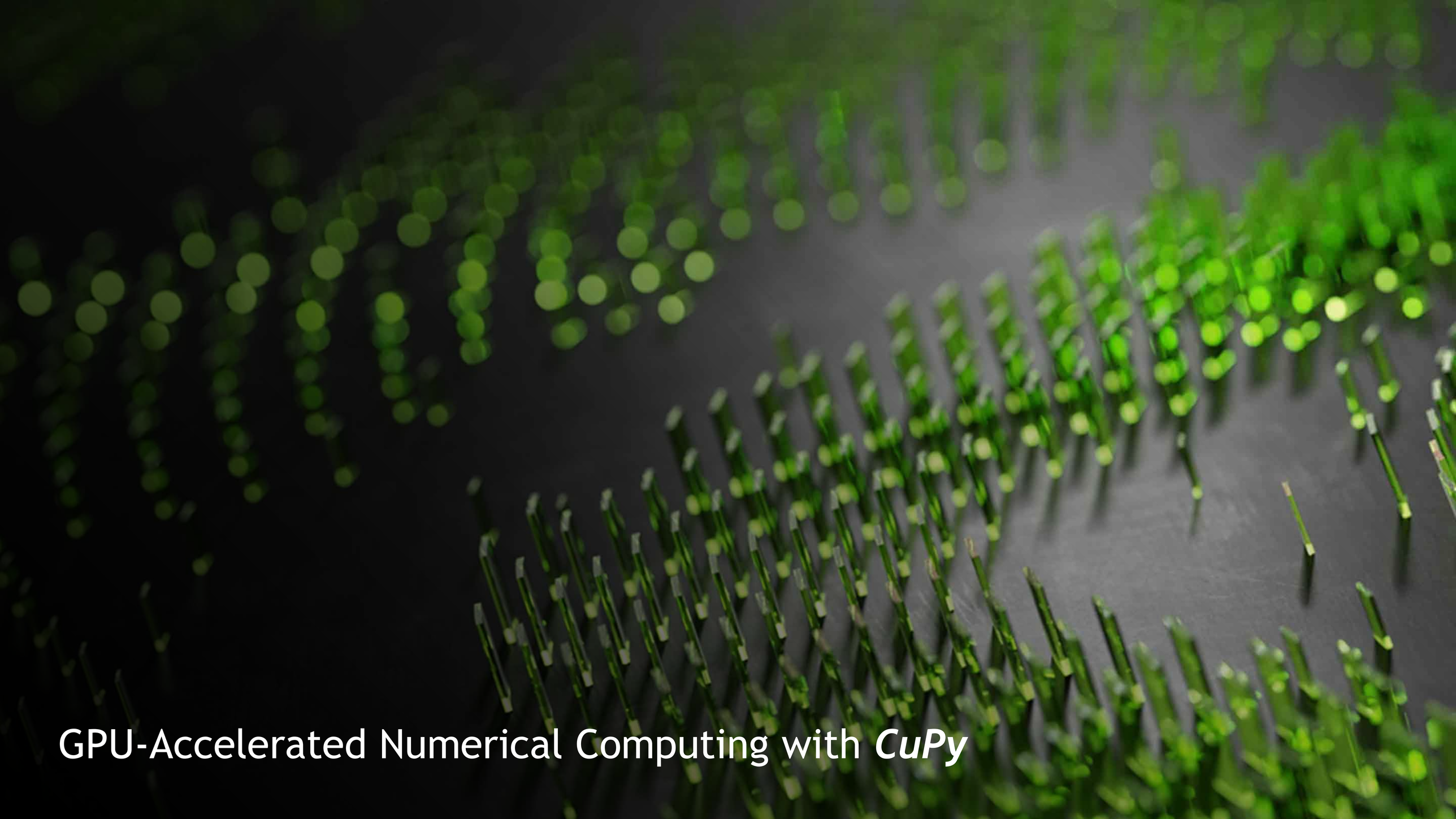
- There is a rich ecosystem of GPU-accelerated libraries *https://developer.nvidia.com/gpu-accelerated-libraries*
- Profiling tools (Nsight) are compatible with Python GPU tools

We care about performance - make a relevant test suite!

- Many applications are already GPU-accelerated
- https://www.nvidia.com/en-us/gpu-accelerated-applications/
- https://ngc.nvidia.com/

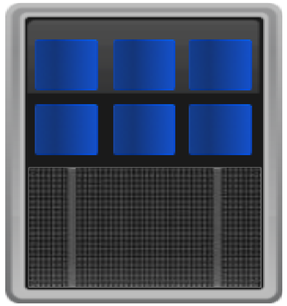
PYTHON GPU ECOSYSTEM

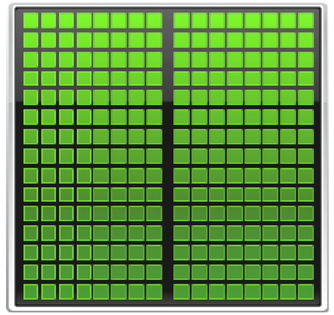




GPU-Accelerated Numerical Computing with *CuPy*

NUMERICAL COMPUTING IN PYTHON





- Mathematical focus
- Operates on arrays of data
  - *ndarray*, holds data of same type
- Many years of development
- Highly tuned for CPUs

- NumPy like interface
- Trivially port code to GPU
- Copy data to GPU
  - CuPy *ndarray*
- Data interoperability with DL frameworks, RAPIDS, and Numba
- Uses high tuned NVIDIA libraries
- Can write custom CUDA functions

CUPY

A NumPy like interface to GPU-acceleration ND-Array operations

BEFORE

```
import numpy as np

size = 4096
A = np.random.randn(size,size)

Q, R = np.lingalg.qr(A)
```

AFTER

```
import cupy as cp

size = 4096
A = cp.random.randn(size,size)

Q, R = cp.lingalg.qr(A)
```

NumPy

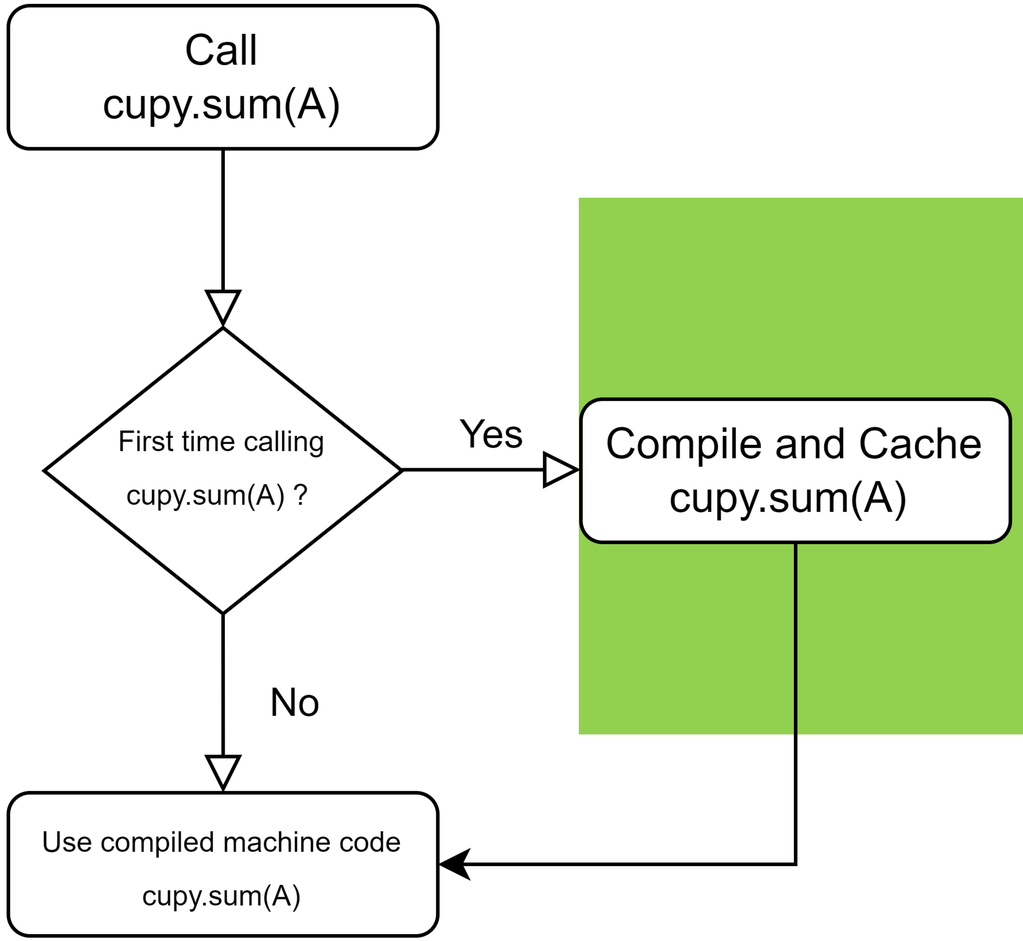
52x Speedup!

CuPy

Notebook 1: Introduction to CuPy

Working in Section:
"Introduction to CuPy"

KERNEL OVERHEAD



JIT Compilation

- What is the size of A?
- What is the datatype?
- Which GPU-accelerated libraries are available?
- Compiler optimizations for custom kernels

KERNEL OVERHEAD

Kernel Launch Overhead

- Python wrappers around CUDA API call for kernel execution
- Kicking off the CUDA kernel (OS, CUDA Driver)
- GPU context switching from another task

- *Cost is on the order of microseconds*

Defensive Strategies

- Increase dataset size
- Combine small kernels together
  - @cupy.fuse for elementwise or reduction kernels
  - Dynamic programming
- Use CUDA streams to backfill GPU work queue
  - Available via CuPy, Numba

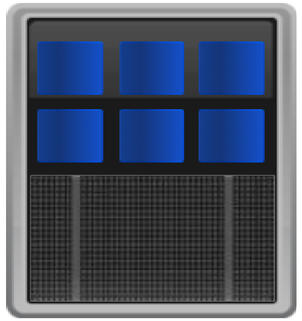
Python Decorators

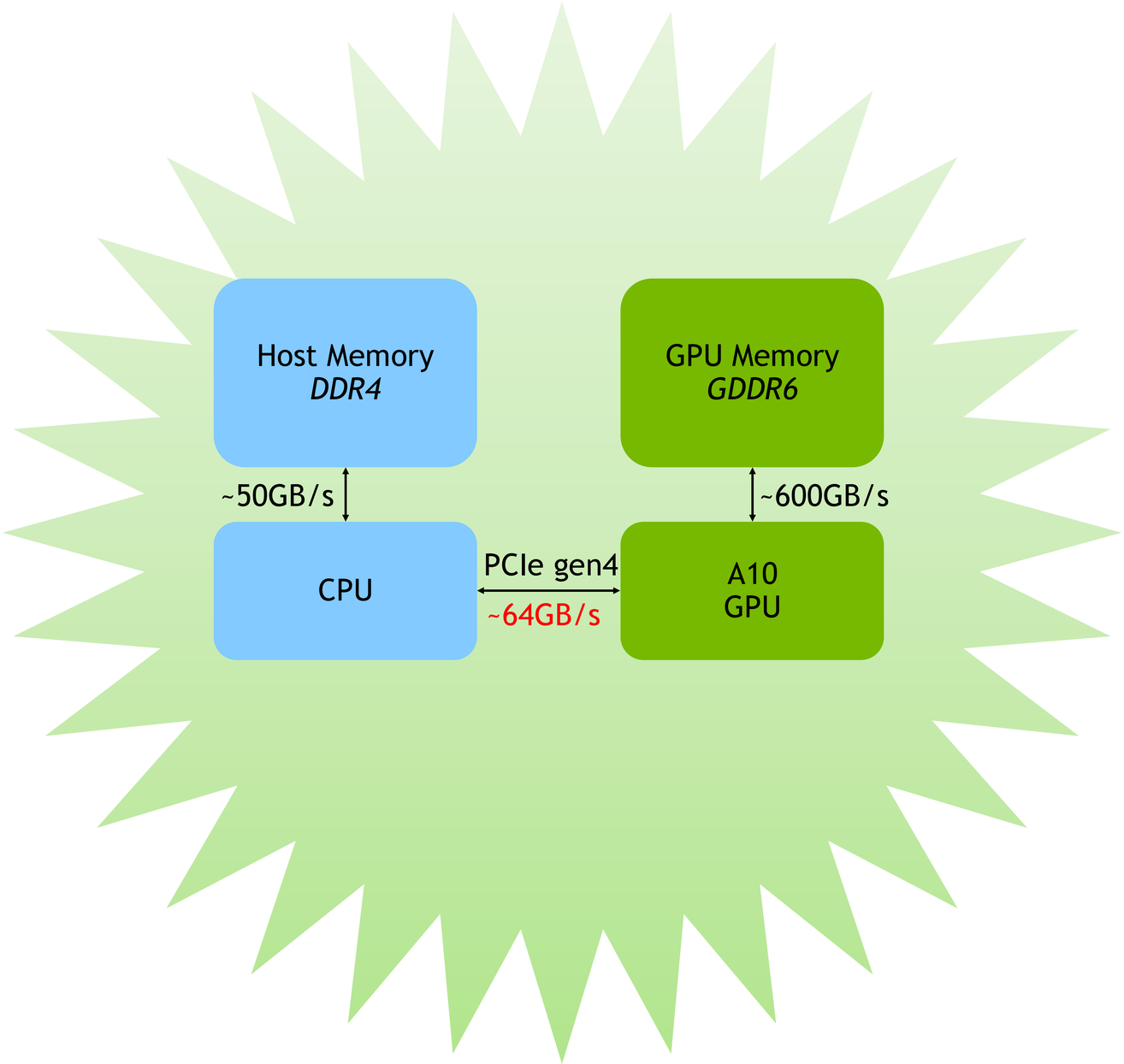
```
@cp.fuse
def fused_squared_diff(x, y):
    return (x - y) * (x - y)


@jit
def Add(a, b):
    return a + b

==================================

#pragma acc parallel
{
    #pragma acc loop
    for(int i= 0; j < N; i++) {
        a[i] = 0;
    }
}
```

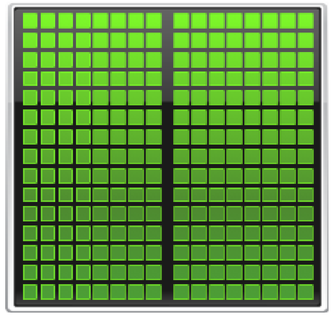
Notebook 1: Introduction to CuPy

Working in Section:
"Kernel Overhead"

DATA MOVEMENT OVERHEAD



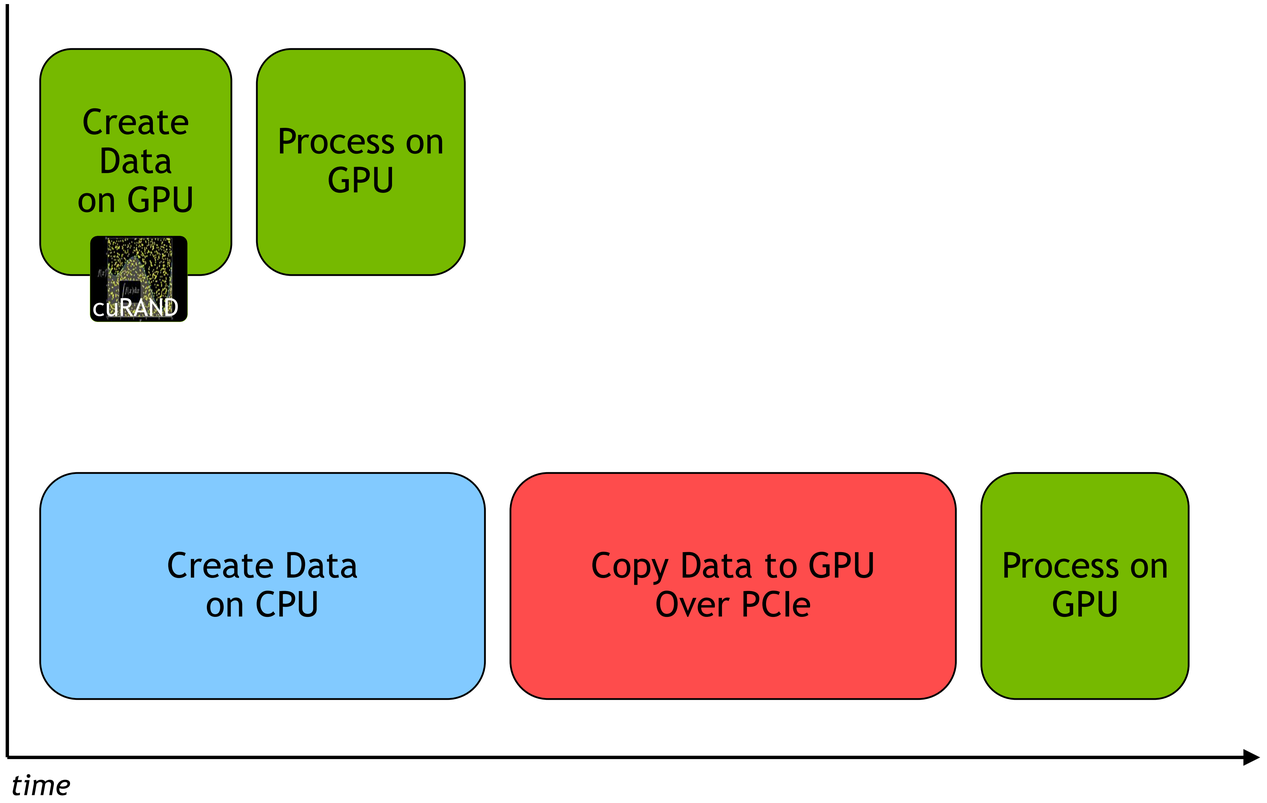




DATA MOVEMENT OVERHEAD

Data Movement Considerations

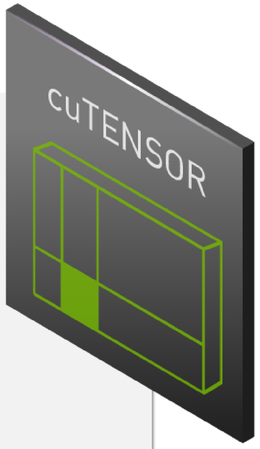
- Do we have enough work to amortize data transfer cost?
- Can we create our data on the GPU instead?
- CuPy ndarrays stay on GPU until retrieved
  - $Y = AX + XA^T$
  - Can we move more of the workflow to GPU?
- Overlap computation with data movement?



Notebook 1: Introduction to CuPy

Working in Section:
"Data Movement Overhead"

CUPY BACKENDS



CUDA: cuTensor

- Tensor linear algebra library for tensor contraction, reduction, and elementwise operations.
- Harness the specialized tensor cores on NVIDIA GPUs for high-performance tensor computations and accelerate deep learning training and inference, computer vision, quantum chemistry, and computational physics workloads.

CuPy: *CUPY ACCELERATORS*

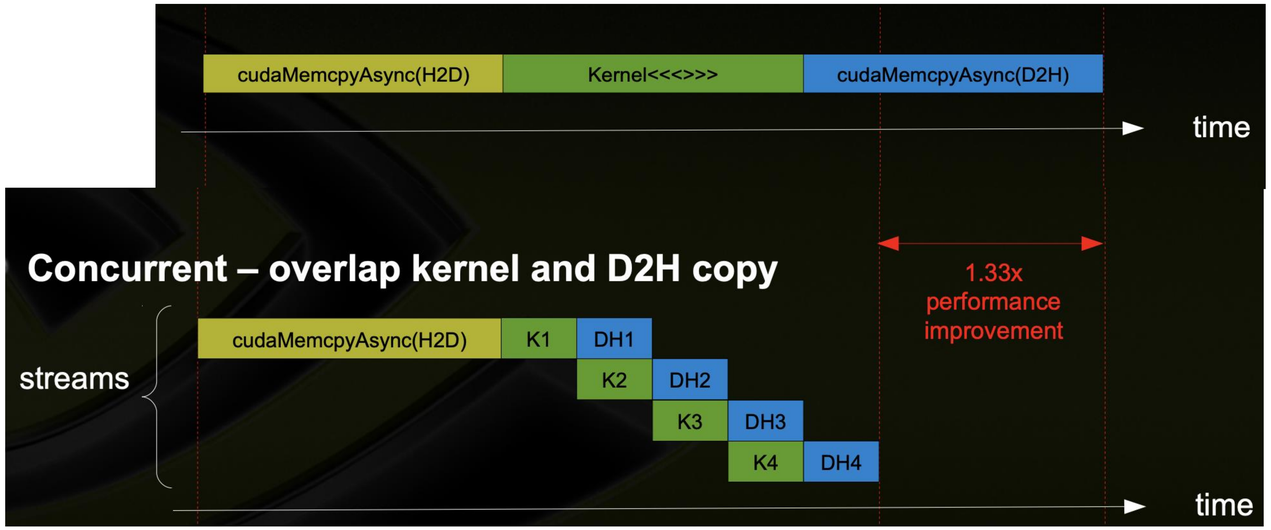
- Makes CUDA implementations available within CuPy when relevant
- cub: cooperative threadblock primitives and other utilities for CUDA kernel programming
- cutensor: high-performance tensor operations
- cutensornet: extension of cuTENSOR focused on tensor network algorithms

Notebook 1: Introduction to CuPy

Working in Section:
"CuPy Backends"

STREAMS AND EVENTS

- **Asynchronous Execution:** Operations are executed in parallel, reducing idle time.
- **Efficient Resource Utilization:** Maximizes GPU usage by overlapping data transfers and computations.
- **Improved Responsiveness:** Enhances system responsiveness by allowing other tasks to proceed without waiting for previous operations to complete.



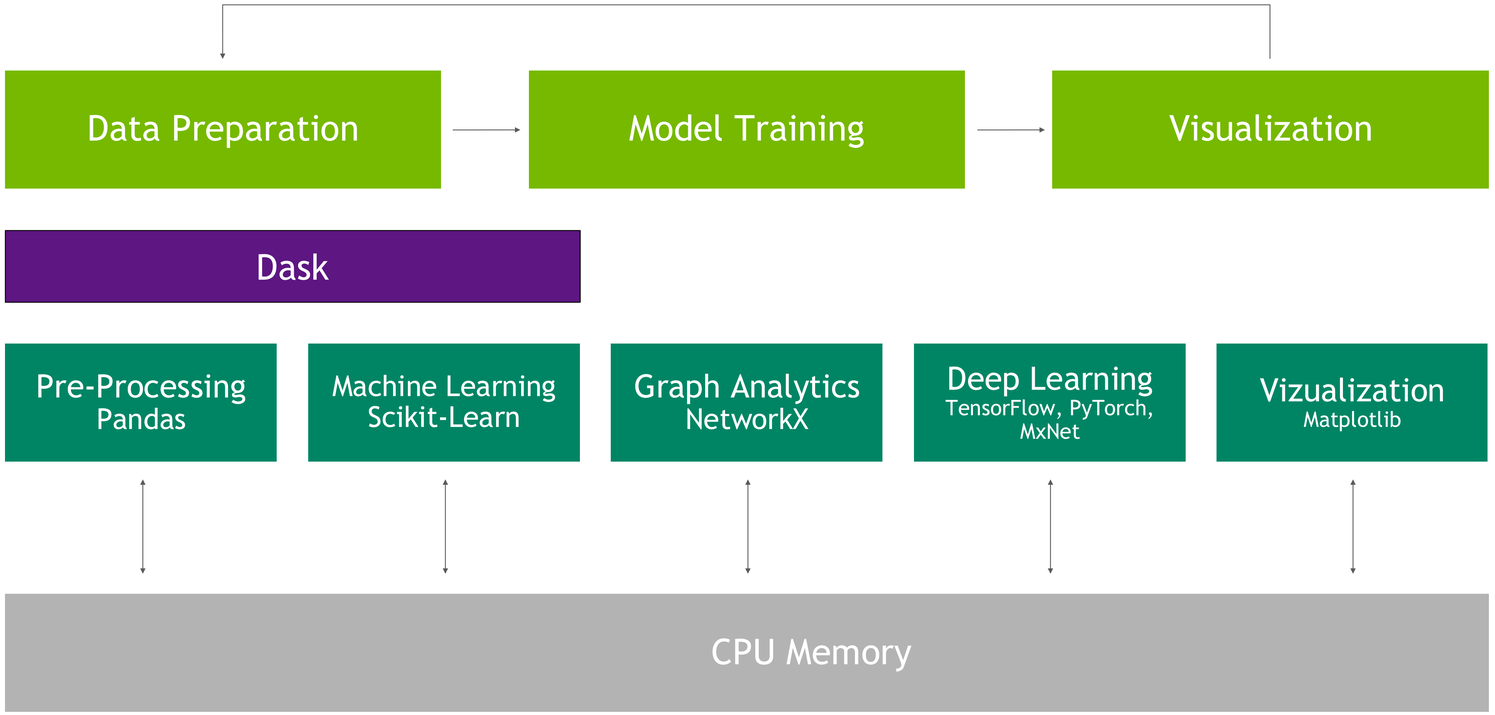
Notebook 1: Introduction to CuPy

Working in Section:
“Streams and Events”

*****Don’t forget to restart the kernel*****

GPU-Accelerated Data Manipulation with *RAPIDS cuDF*

TRADITIONAL DATA SCIENCE APPLICATIONS



RAPIDS PIPELINE

Data Preparation/ETL

cuDF

- GPU-accelerated ETL functions
- Tracks Pandas and other common PyData APIs
- Dask + UCX integration for scaling

RAPIDS for Apache Spark

- RAPIDS accelerator for Apache Spark

Analytics/ML/Graph

RAPIDS ML

- GPU-native cuML library, plus XGBoost, RAFT, FIL, HPO, and more

cuGraph

- GPU graph analytics, including Louvain, PageRank, and more
- Multi-Node Multi-GPU features

Visualization

cuxfilter

- GPU-accelerated cross-filtering

Viz integration

- pyViz: Plotly Dash, Bokeh, Datashader, HoloViews, hvPlot
- Node-RAPIDS bindings for node.js

Application and Domain-Specific Frameworks

Morpheus
Cybersecurity application development framework

Merlin
Recommender Systems development framework

cuSignal
Signals processing

cuCIM
Computer vision & image processing primitives

cuSpatial
Spatial analytics

NVTabular
Tabular data feature engineering

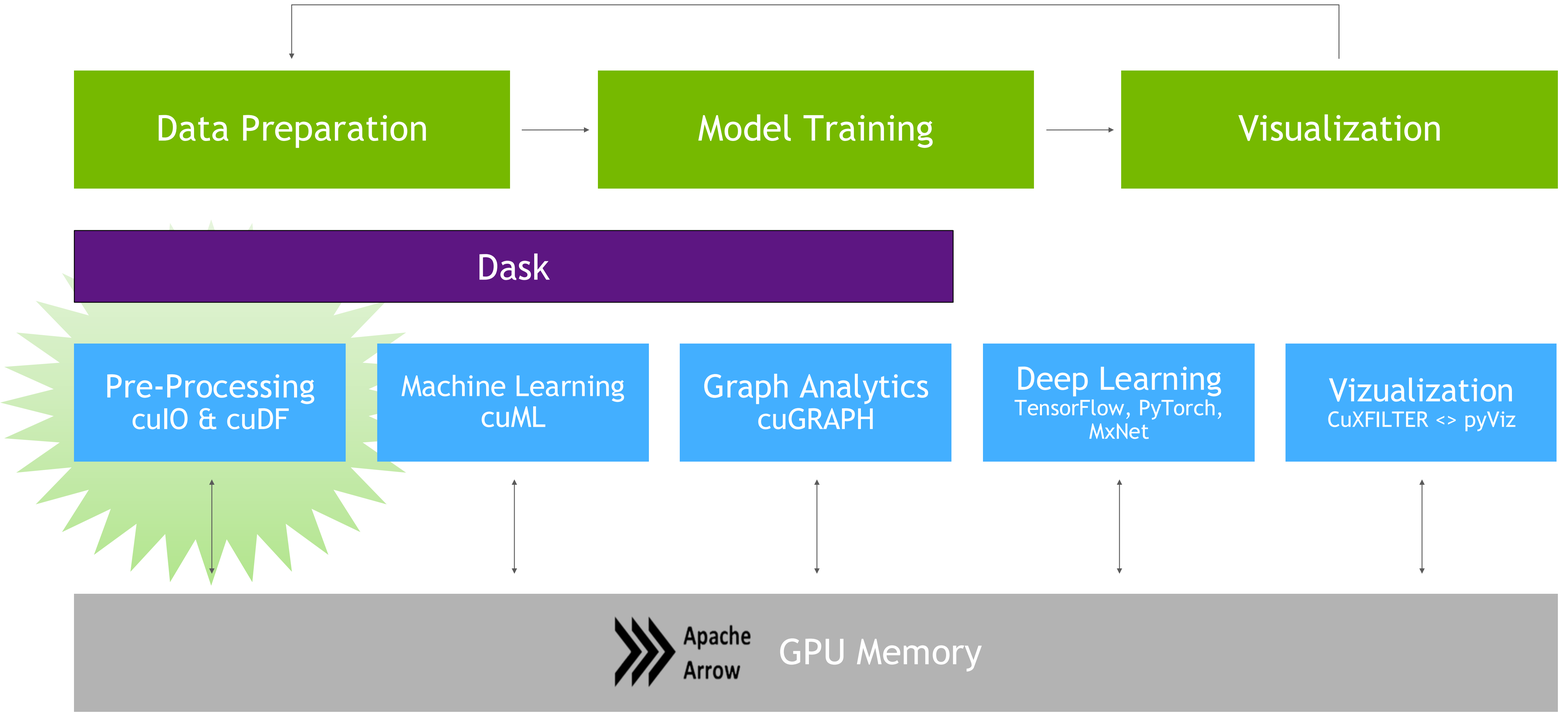
DATA SCIENCE API ALIGNMENT

Open source software that accelerates popular data science packages

Function	CPU	GPU/RAPIDS
Data handling	pandas	cuDF **
Machine learning	scikit-learn	cuML **
Graph analytics	NetworkX	cuGraph
Geospatial	GeoPandas/SciPy	cuSpatial
Signals	SciPy.signal	cuSignal
Image Processing	scikit-image	cuCIM

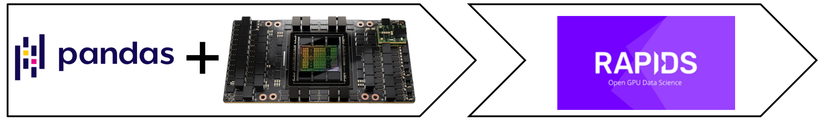
The RAPIDS and GPU-accelerated PyData stack bring GPGPU to data scientists at the Python layer providing familiar APIs without the steep curve of learning new programming language or paradigm

RAPIDS: GPU-ACCELERATED DATA SCIENCE
WITH API ALIGNMENT



GPU-ACCELERATED PANDAS WITH CUDF

- Use RAPIDS CuDF to accelerate computationally expensive ETL operations
- Manipulate GPU DataFrames following the Pandas API
- Create GPU DataFrames from Numpy arrays, CuPy arrays, Pandas DataFrames, and PyArrow Tables
- Python interface to CUDA C++ library with additional functionality
- Available via pip and conda

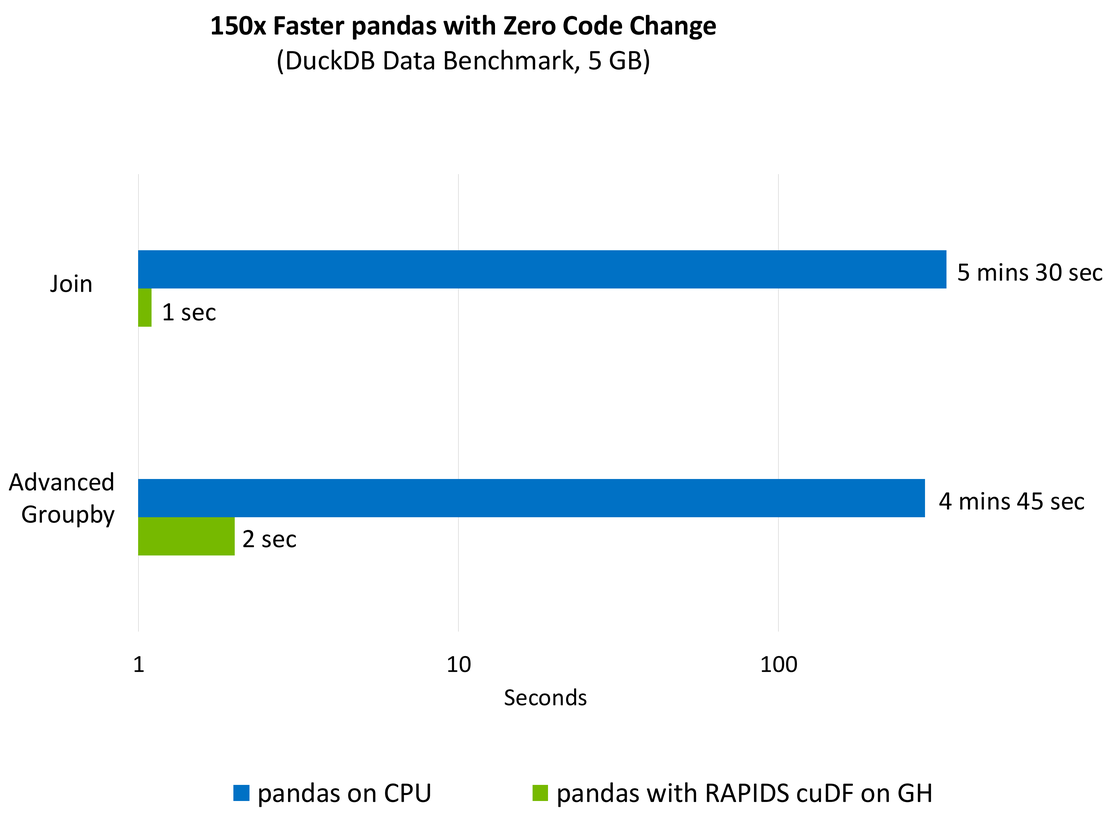



```
import cudf as pd
import numpy as np
from time import time

import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

wine_set = pd.read_csv("data/winequality.csv")

wine_set.head(n=5)
wine_set.tail(n=5)
```

RAPIDS CUDF

World's Fastest Data Analytics with pandas

- 150x Faster than CPU-only
- Unified workflow on CPUs and GPUs across laptops, workstation & datacenter
- Compatible with third-party libraries built on pandas



NVIDIA Grace Hopper vs. Intel Xeon Platinum 8480CL CPU

Notebook 2: Introduction to RAPIDS cuDF

Working in Section:
"GPU-Accelerated Data Manipulation with CuDF"

311 DATASET EXAMPLE

311 Service Requests from 2010 to Present

Social Services

NOTE: Data after Saturday, 1/25 may not be included in this dataset. We are currently working on fixing this issue.

...

Read more

Last Updated
February 7, 2025

Data Provided By
311

About this Dataset

Updated
February 7, 2025

Data Last Updated	Metadata Last Updated
February 7, 2025	February 6, 2025

Date Created
October 9, 2011

Views	Downloads
937K	449K

Data Provided by	Dataset Owner
311	NYC OpenData

Dataset Information

Agency	Office of Technology and Innovation (OTI)

Update

Update Frequency	Daily
Automation	Yes
Date Made Public	10/18/2011

Attachments

311_ServiceRequest_2010-Present_DataDictionary_Updated_2023.xlsx

Topics

Category	Social Services
Tags	311, 311 service requests, city government, socialservices, service request status, all service requests, rodent, rat, bike, pothole, complaints

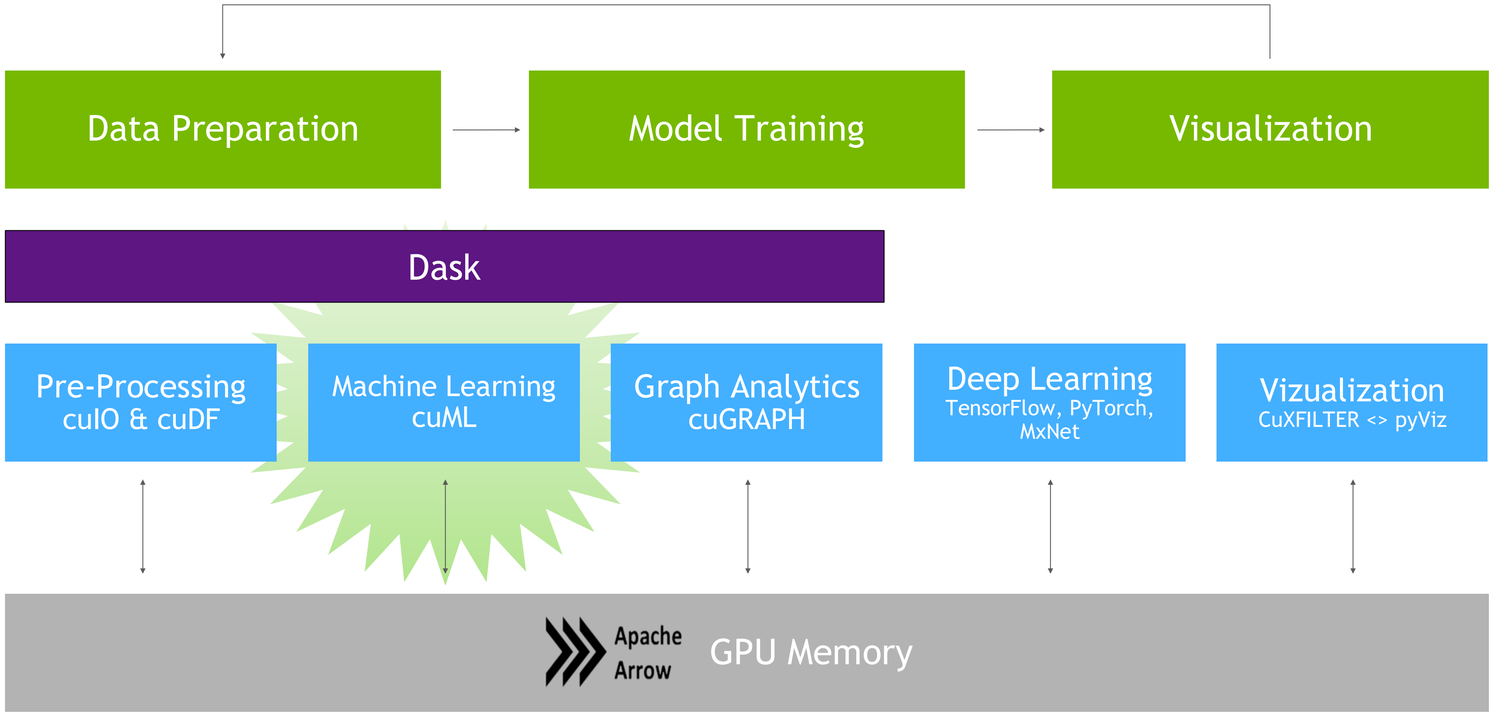
Notebook 2: Introduction to RAPIDS cuDF

Working in Section:
“CuDF pandas in Practice: 311 call dataset”

<u>**Don’t forget to restart the kernel**</u>

GPU-Accelerated Data Science with *RAPIDS cuML*

RAPIDS: GPU-ACCELERATED DATA SCIENCE
WITH API ALIGNMENT



CUML

Accelerated Machine Learning with a Scikit-Learn API

CPU

Scikit-learn

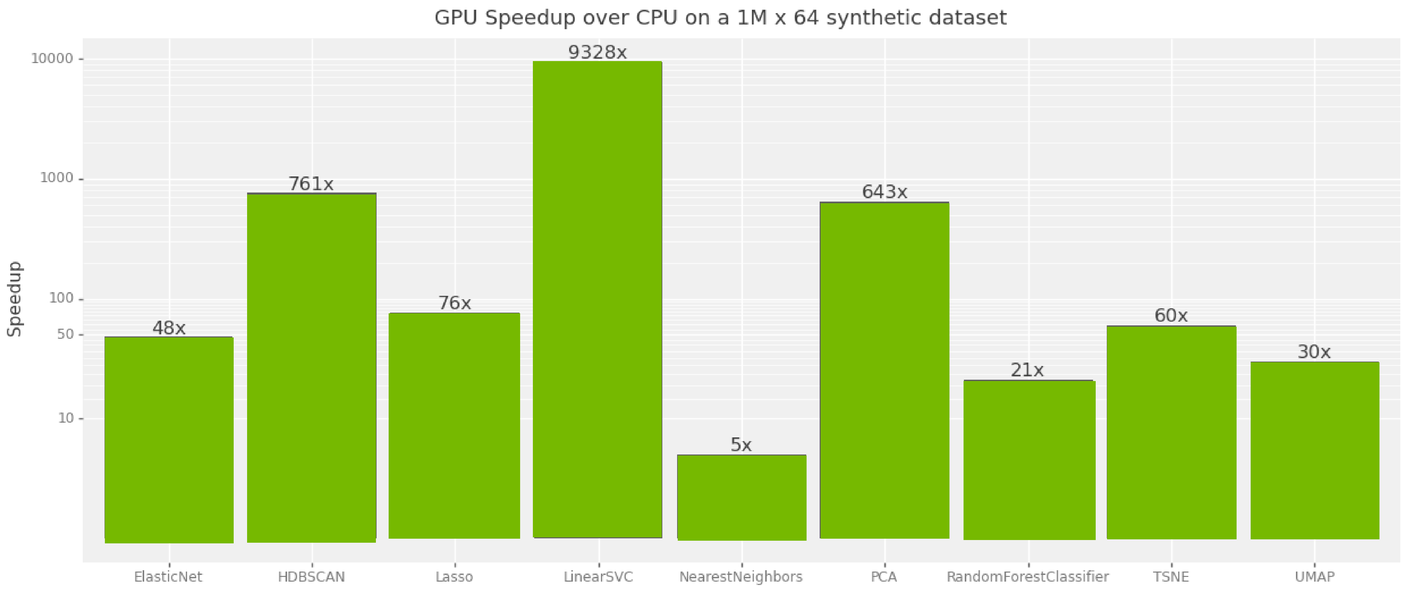
```
>>> from sklearn.ensemble import
RandomForestClassifier
>>> clf = RandomForestClassifier()
>>> clf.fit(x, y)
```

GPU

cuML

```
>>> from cuml.ensemble import
RandomForestClassifier
>>> clf = RandomForestClassifier()
>>> clf.fit(x, y)
```

50+ GPU-Accelerated Algorithms & Growing



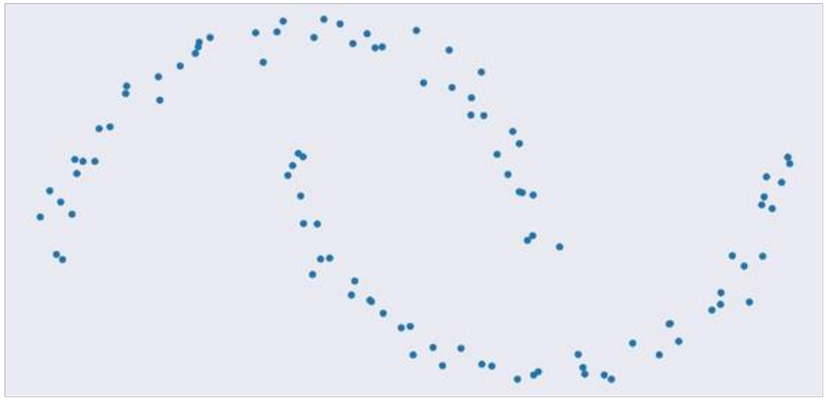
Time Series	Classification	Regression	Clustering	Preprocessing
Cross Validation	Tree Models	Dimensionality Reduction		Explainability

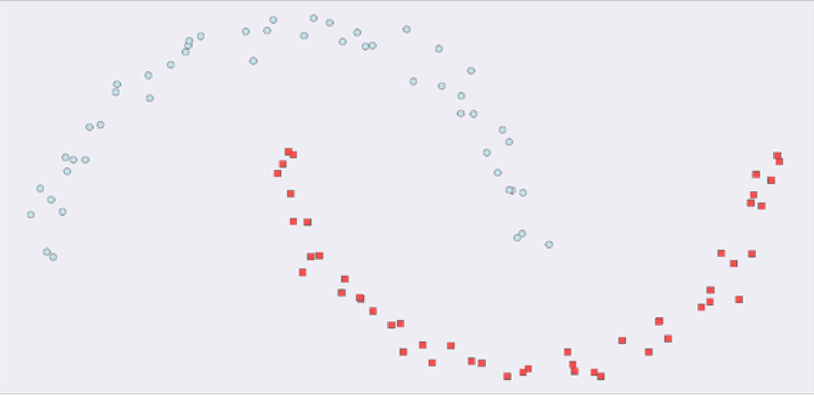
A100 GPU vs. AMD EPYC 7642 (96 logical cores)
cuML 23.04, scikit-learn 1.2.2, umap-learn 0.5.3

DATA SCIENCE IN PYTHON

```
from sklearn.datasets import make_moons
import pandas


X, y = make_moons(n_samples=int(1e2),
                  noise=0.05, random_state=0)

X = pandas.DataFrame({'fea%d'%i: X[:, i]
           for i in range(X.shape[1])})
```

```
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps = 0.3, min_samples = 5)


y_hat = dbscan.fit_predict(X)
```

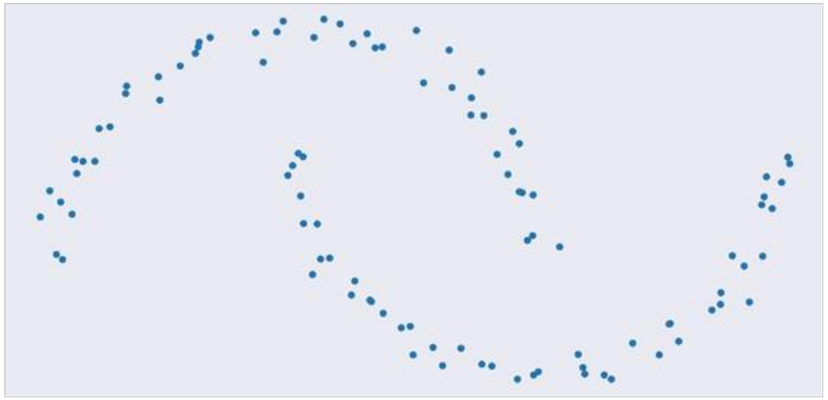


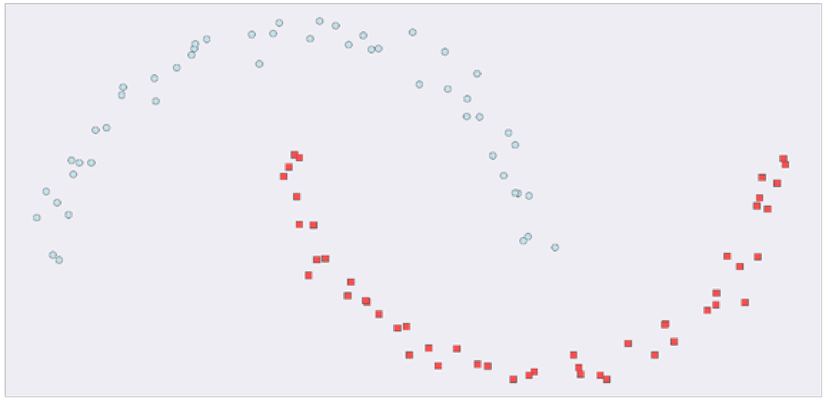


DATA SCIENCE IN PYTHON

```
from sklearn.datasets import make_moons
import cudf

X, y = make_moons(n_samples=int(1e2),
            noise=0.05, random_state=0)

X = cudf.DataFrame({'fea%d'%i: X[:, i]
            for i in range(X.shape[1])})
```

```
from cuml import DBSCAN
dbscan = DBSCAN(eps = 0.3, min_samples = 5)

y_hat = dbscan.fit_predict(X)
```





Notebook 3: Introduction to RAPIDS

Working in Section:
“GPU-Accelerated Machine Learning with CuML”

<u>**Don’t forget to restart the kernel**</u>

311 DATASET EXAMPLE

Rows	Columns	Each row is a
39M	41	311 Service Request

Columns (41)

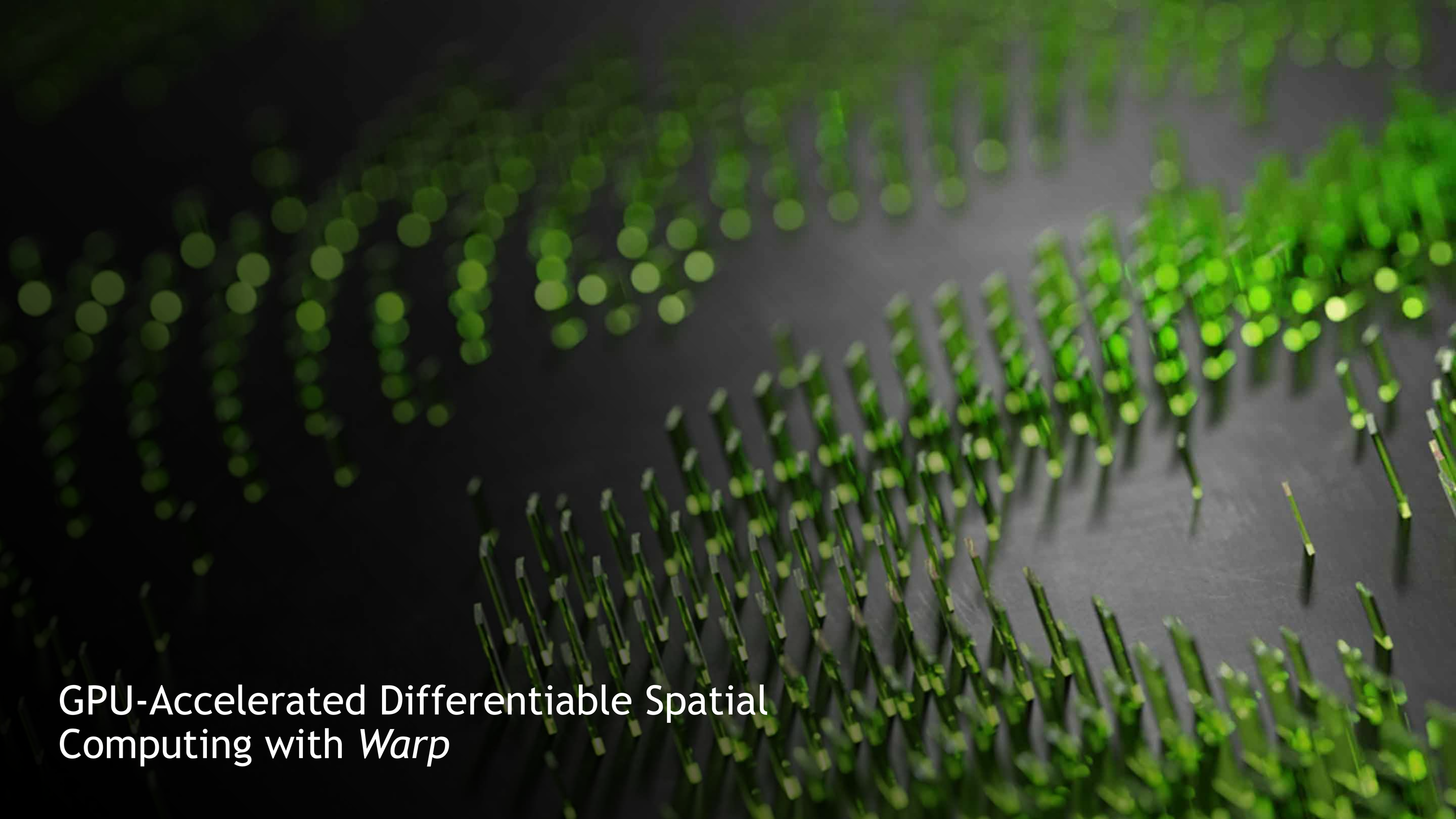
Column Name	Description	API Field Name	Data Type
Unique Key	Unique identifier of a Service Request (SR) in the open data set	unique_key	Text
Created Date	Date SR was created	created_date	Floating Timestamp
Closed Date	Date SR was closed by responding agency	closed_date	Floating Timestamp
Agency	Acronym of responding City Government Agency	agency	Text
Agency Name	Full Agency name of responding City Government Agency	agency_name	Text
Complaint Type	This is the first level of a hierarchy identifying the topic of the incident or condition. Complaint Type may have a corresponding Descriptor (below) or may stand alone.	complaint_type	Text
Descriptor	This is associated to the Complaint Type, and provides further detail on the incident or condition. Descriptor values are dependent on the Complaint Type, and are not always required in SR.	descriptor	Text
Location Type	Describes the type of location used in the address information	location_type	Text
Incident Zip	Incident location zip code, provided by geo validation.	incident_zip	Text
Incident Address	House number of incident address provided by submitter.	incident_address	Text
Street Name	Street name of incident address provided by the submitter	street_name	Text
Cross Street 1	First Cross street based on the geo validated incident location	cross_street_1	Text
Cross Street 2	Second Cross Street based on the geo validated incident location	cross_street_2	Text
Intersection Street 1	First intersecting street based on geo validated incident location	intersection_street_1	Text
Intersection Street 2	Second intersecting street based on geo validated incident location	intersection_street_2	Text

Rows per page: 15 1-15 of 41

Notebook 3: Introduction to RAPIDS cuML

Working in Section:
“CuML pandas in Practice: 311 call dataset”

<u>**Don’t forget to restart the kernel**</u>

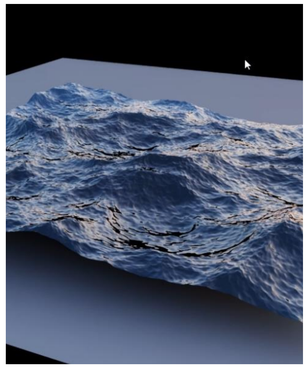
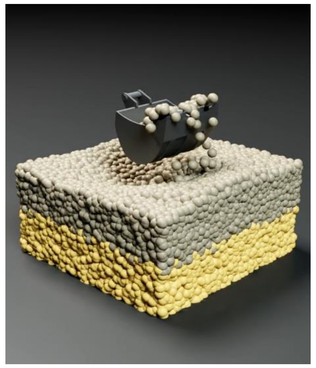


GPU-Accelerated Differentiable Spatial Computing with *Warp*

WARP

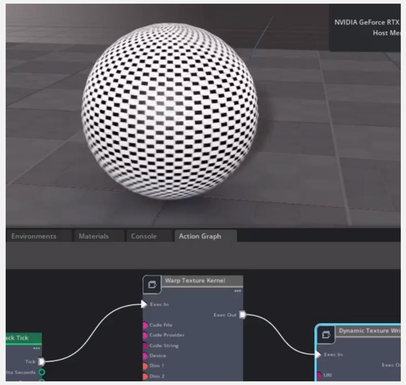
- Make it easy to write GPU simulation code from Python
- Lower barrier to entry, fast iteration time
- Built-in data structures and algorithms for geometry processing
- Differentiable for fast training
- Efficient interop with NumPy, CuPy, PyTorch, JAX, and existing C++/CUDA code

```
@wp.kernel
def integrate(x: wp.array(dtype=wp.vec3),
              v: wp.array(dtype=wp.vec3),
              f: wp.array(dtype=wp.vec3),
              w: wp.array(dtype=float),
              gravity: wp.vec3,
              dt: float):

    # thread id
    tid = wp.tid()

    # semi-implicit Euler step
    v[tid] = v[tid] + (f[tid] * w[tid] + gravity) * dt
    x[tid] = x[tid] + v[tid] * dt


# kernel launch
wp.launch(integrate, dim=N, inputs=[...], device="cuda")
```

WARP USE CASES

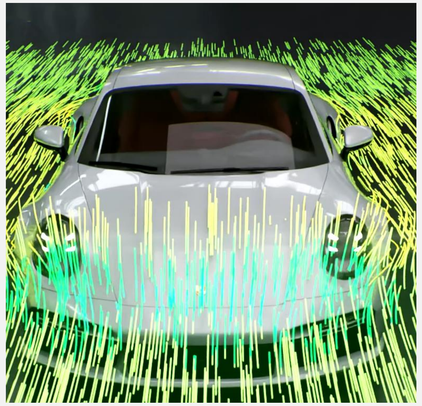
Geometry Processing



Examples:

- Mesh distance/sampling queries (Modulus)
- SDF generation (NanoVDB)
- Point-cloud processing
- GNN construction (Modulus)
- Visualization and analysis of CFD data

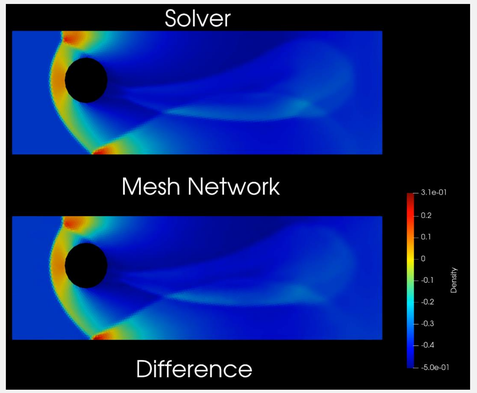
Differentiable Simulation



Examples:

- Efficient lattice-Boltzmann kernels (XLB)
- Accelerated robot policy learning (SHAC)
- Neural reduced-order methods (MIT)
- Physics NeRF (UToT)
- Accelerated FEM methods

Accelerated Models and Training



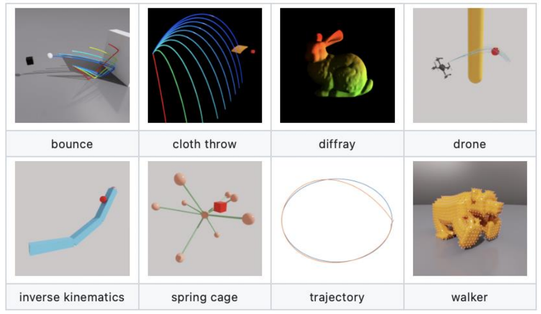
Examples:

- Image processing
- Memory-efficient ConvNet
- Accelerated RFdiffusion
- Accelerated equivariant neural networks
- Accelerated KAN networks

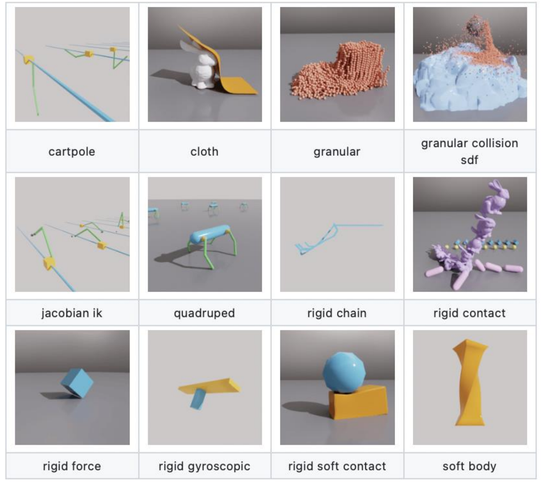
WARP PYTHON MODULES

- warp (core library)
  - Differentiable kernel coding for Python
  - Math, geometry, vector library
- warp.sim
  - Differentiable real-time simulation for robotic control + prediction
  - Rigid bodies, soft bodies, particles, cloth
  - URDF, MJCF, UsdPhysics parsers
- warp.fem
  - Differentiable PDE framework for heat transfer, diffusion, elasticity, etc.
  - Easily change the underlying discretization type without rewriting code

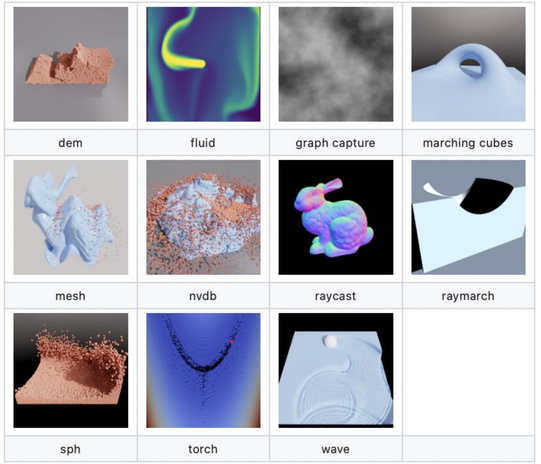
examples/optim



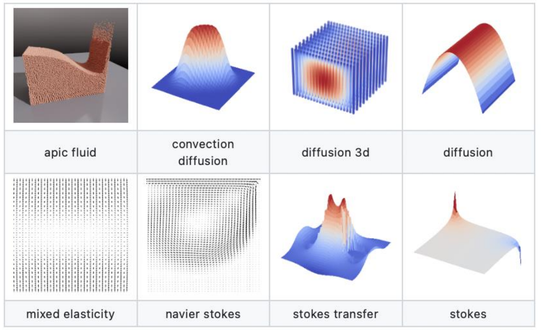
examples/sim



examples/core



examples/fem



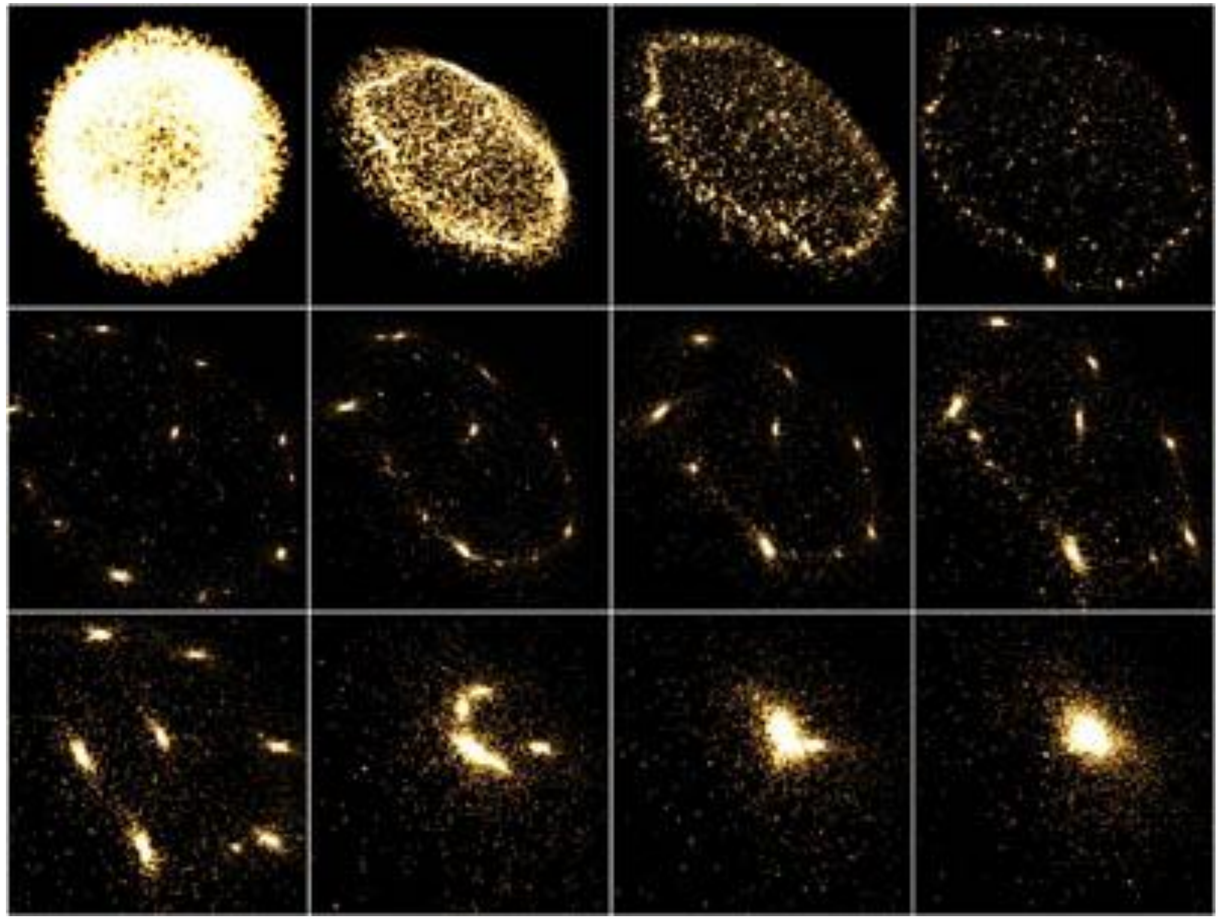
Notebook 4: Introduction to Warp

Working in Section:
"Warp Core"

WARP N BODY SIMULATION

- A computational method used to model the motion of a system of N objects, such as:
  - planets
  - stars
  - galaxies, under the influence of their mutual gravitational interactions.

- Understand how systems evolve over time and provide insights into phenomena like gravitational waves and the distribution of matter in the cosmos



Frames from an Interactive 3D Rendering of a 16,384-Body System Simulated using CUDA

Notebook 4: Introduction to Warp

Working in Section:
"N-body Sample"

WARP FEM

Differentiable framework for PDEs

- Flexible finite element-based (FEM/DG) framework for:
  - Diffusion
  - Convection
  - Fluid flow
  - Elasticity
- Define weak-form PDE in high-level Python syntax
  - wp.fem.grad()
  - wp.fem.div()
  - wp.fem.curl()
- Combine with geometry + shape functions
  - wp.fem.TetMesh()
  - wp.fem.HexMesh()
  - wp.fem.make_polynomial_space()
- Integrate to assemble linear system
  - wp.fem.integrate()
- Solve with block-sparse (BSR) solvers
  - warp.sparse.BsrMatrix

Define PDE

```
import warp.fem as fem

@fem.integrand
def diffusion_form(s: fem.Sample, u: fem.Field, v: fem.Field, nu: float):
    return nu * wp.dot(
        fem.grad(u, s),
        fem.grad(v, s),
)
```

Define Geometry

```
geo = fem.Tetmesh(positions=positions, tet_vertex_indices=indices)
scalar_space = fem.make_polynomial_space(
    geo, degree=2, element_basis=fem.ElementBasis.SERENDIPITY)
```

Assemble System

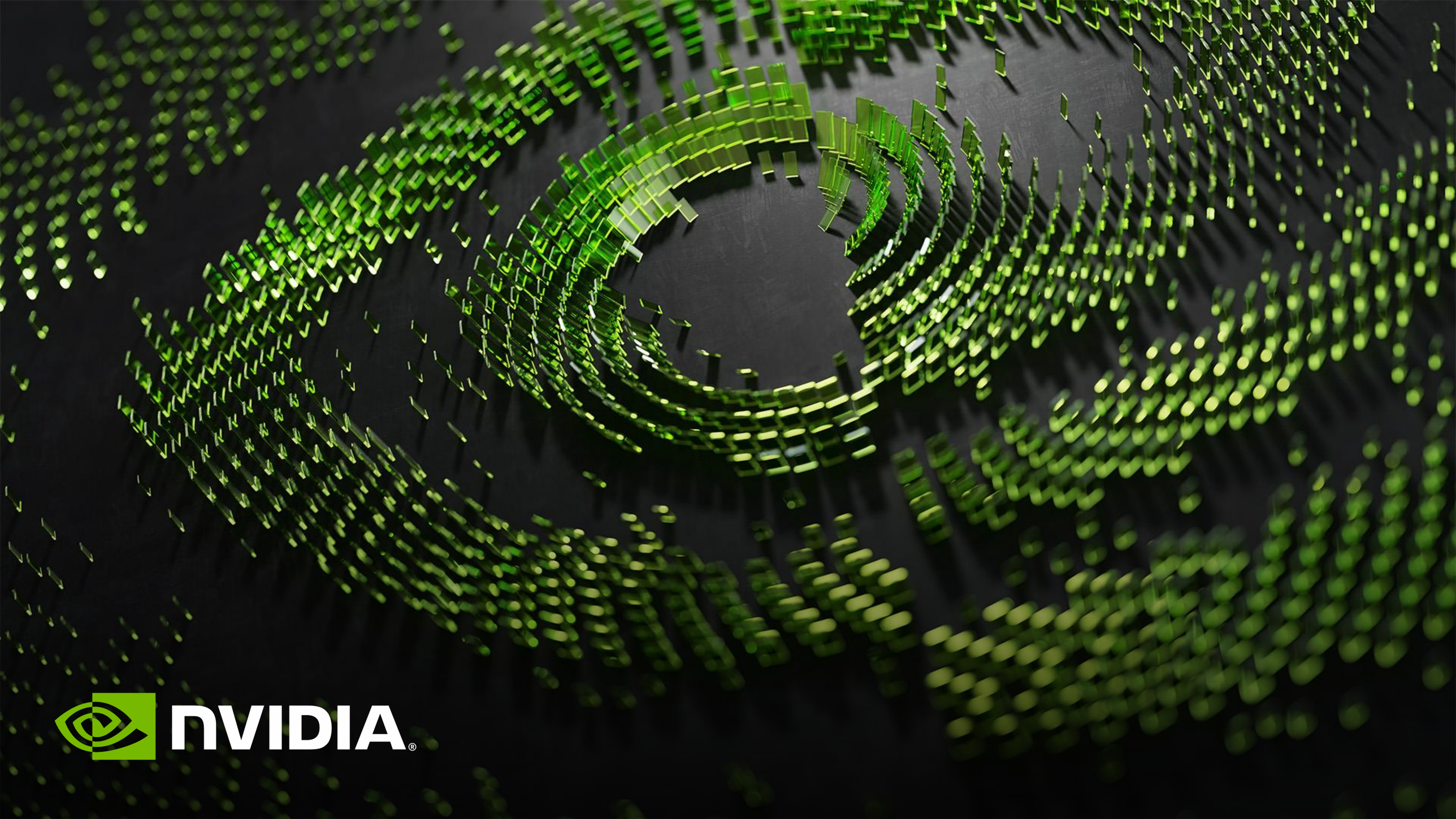
```
trial = fem.make_trial(scalar_space)
test = fem.make_test(scalar_space)

matrix = fem.integrate(diffusion_form,
            fields={"u": trial, "v": test},
            values={"nu": 0.1})
```

Example: Diffusion PDE bilinear form

Notebook 4: Introduction to Warp

Working in Section:
“Finite Element Modeling Example”

*****Don’t forget to restart the kernel*****


NVIDIA